Yasmin Samy

lec 1

Ch 6

Sheet #6 – Co

Computer Architecture

Sheet (6)

6.1 Consider the binary numbers in the following addition and subtraction problems to be signed, 6-bit values in the 2's-complement representation. Perform the operations indicated, specify whether or not arithmetic overflow occurs, and check your answers by converting operands and results to decimal sign-and-magnitude representation.

```
(a)   110111    010101  (d)
     +111001   +101011

(b)   111110    100001  (e)
     -100101   -011101

(c)   000111    011010  (f)
     -111000   -100010
```

6.9 Show that the logic expression $c_n \oplus c_{n-1}$ is a correct indicator of overflow in the addition of 2's-complement integers, by using an appropriate truth table.

6.10 (*a*) Design a 64-bit adder that uses four of the 16-bit carry-lookahead adders shown in Figure 6.5 along with additional logic to generate c_{16}, c_{32}, c_{48}, and c_{64}, from c_0 and the G_i^{II} and P_i^{II} variables shown in this figure. What is the relationship of the additional logic to the logic inside each lookahead circuit in the figure?

Figure 6.5

(*b*) Show that the delay through the 64-bit adder is 12 gate delays for $\overset{\text{sum}}{s_{63}}$ and 7 gate delays for $\overset{\text{Carry}}{c_{64}}$, as claimed at the end of Section 6.2.1.

(*c*) Compare the gate delays to produce $\boxed{s_{31} \text{ and } c_{32}}$ in the 64-bit adder of part (*a*) to the gate delays for the same variables in the 32-bit adder built from a cascade of two 16-bit adders, as discussed in Section 6.2.1.

6.11 (*a*) How many logic gates are needed to build the 4-bit carry-lookahead adder shown in Figure 6.4?

(*b*) Use appropriate parts of the result from Part (*a*) to calculate how many logic gates are needed to build the 16-bit carry-lookahead adder shown in Figure 6.5.

6.12 Show that the worst case delay through an $n \times n$ array of the type shown in Figure 6.6*b* is $6(n-1)-1$ gate delays, as claimed in Section 6.3.

Figure 6.6b

6.17 Multiply each of the following pairs of signed 2's-complement numbers using the Booth algorithm. In each case, assume that A is the multiplicand and B is the multiplier.

(*a*) $A = 010111$ and $B = 110110$

(*b*) $A = 110011$ and $B = 101100$

(*c*) $A = 110101$ and $B = 011011$

(*d*) $A = 001111$ and $B = 001111$

6.18 Repeat Problem 6.17 using bit-pairing of the multipliers.

6.19 Indicate generally how to modify the circuit diagram in Figure 6.7*a* to implement multiplication of signed, 2's-complement, n-bit numbers using the Booth algorithm, by clearly specifying inputs and outputs for the Control sequencer and any other changes needed around the adder and A register.

*) Using non-restoring division perform the operation A % B on the five bit numbers A=10101 and B=00101.

Sheet # 6
Solution

لو (1) ← سالب
(0) ← موجب

Chapter 6 – Arithmetic

6-bit signed

answer

6.1. Overflow cases are specifically indicated. In all other cases, no overflow occurs.

```
  010110     (+22)     101011     (−21)     111111      (−1)
+ 001001    + (+9)   + 100101   + (−27)   + 000111     + (+7)
  011111     (+31)     010000     (−48)     000110      (+6)
                       overflow

  011001     (+25)  a) 110111      (−9)  d) 010101     (+21)
+ 010000   + (+16)   + 111001    + (−7)   + 101011   + (−21)
  101001     (+41)     110000     (−16)     000000       (0)
  overflow

  010110     (+22)     010110
− 011111   − (+31)   + 100001
              (−9)     110111

b)111110      (−2)     111110
− 100101   − (−27)   + 011011
             (+25)     011001

e)100001     (−31)     100001
− 011101   − (+29)   + 100011
             (−60)     000100
                       overflow

  111111      (−1)     111111
− 000111    − (+7)   + 111001
              (−8)     111000

c)000111      (+7)     000111
− 111000    − (−8)   + 001000
             (+15)     001111

f)011010     (+26)     011010
− 100010   − (−30)   + 011110
             (+56)     111000
                       overflow
```

−9 → 9 → $\overset{8\,4\,2\,1}{001001}$ $\xrightarrow{2's}$ $(110111)_{2's}$

−21 → 21 → $\overset{32\,16\,8\,4\,2\,1}{010101}$ $\xrightarrow{2's}$ $(101011)_{2's}$

−31 → 31 → 011111 $\xrightarrow{2's}$ $(100001)_{2's}$

−27 → 27 → $\overset{32\,16\,8\,4\,2\,1}{011011}$ $\xrightarrow{2's}$ $(100101)_{2's}$

+22 → $(\boxed{0}\overset{16\,8\,4\,2\,1}{10110})_{2's}$

+26 → $(\boxed{0}\overset{16\,8\,4\,2\,1}{11010})_{2's}$

سالب $\boxed{1}10000$ $\xrightarrow{2's}$ $0\overset{16\,8\,4\,2\,1}{10000}$ → 16 → −16

سالب $\boxed{1}01011$ $\xrightarrow{2's}$ $\overset{16\,8\,4\,2\,1}{010101}$ → 21 → −21

موجب $\boxed{0}\overset{16\,8\,4\,2\,1}{11001}$ → $+(16+8+1) = +25$

موجب $\boxed{0}\overset{4\,2\,1}{00111}$ → $+7 = (1+2+4)$

Sheet # 6

(6-1)

$$\begin{array}{rr} & 110111 \\ + & 111001 \\ \hline & 110000 \end{array} \qquad \begin{array}{r} (-9) \\ (-7) \\ \hline -16 \end{array} \qquad\qquad \begin{array}{r} 010101 \\ 101011 \\ \hline 000000 \end{array} \qquad \begin{array}{r} (+21) \\ (-21) \\ \hline (0) \end{array}$$

$$\begin{array}{rr} & 111110 \\ - & 100101 \end{array} \qquad \begin{array}{rr} & (-2) \\ - & (-27) \\ \hline & (+25) \end{array}$$

$\Downarrow$

$$\begin{array}{rr} & 111110 \\ 2's + & 011011 \\ \hline * & \underbrace{011001}_{16\ 8\ 4\ 2\ 1} \end{array} \Longrightarrow (16+8+1) \Downarrow (25)_2$$

$+$

$111110 \xrightarrow{2's} 000010_2 \rightarrow 2 \rightarrow (-2)_{10}$

$100101 \xrightarrow{2's} 011011 \rightarrow 16+8+2+1 \ (27) \rightarrow (-27)_2$

$$\begin{array}{rr} & 100001 \\ - & 011101 \end{array}$$

$\Downarrow$

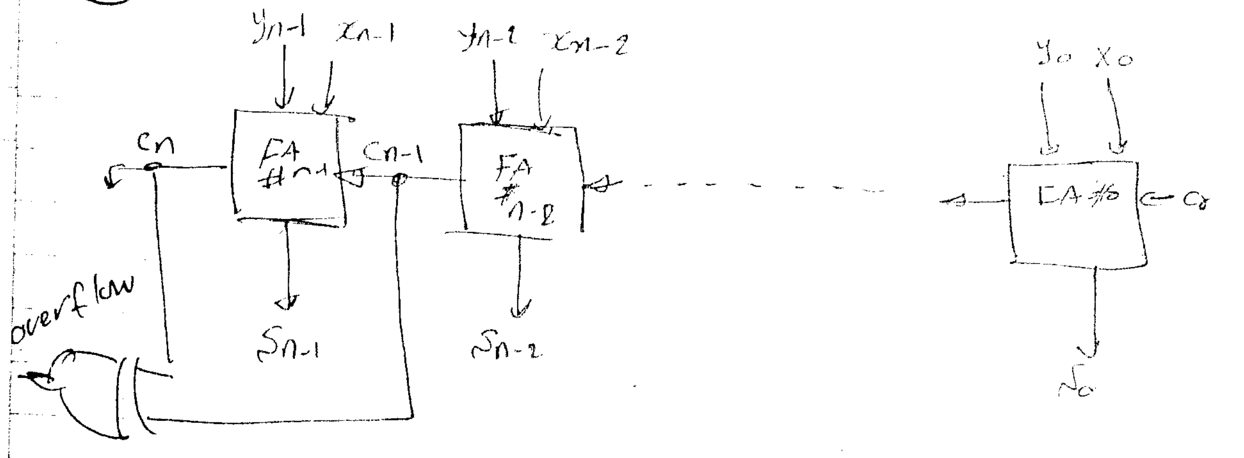
$$\begin{array}{rr} & 100001 \\ + & 100011 \\ \hline * & \underbrace{000100} \end{array} \qquad \begin{array}{rr} & (-31) \\ (2's) & (-29) \\ \hline & (-60)_{10} \end{array}$$

overflow

$100001 \xrightarrow{2's} 011111 \rightarrow 16+8+4+2+1 \rightarrow (31) \rightarrow (-31)$

$100011 \xrightarrow{2's} 011101 \rightarrow 16+8+4+1 = 29 \rightarrow (-29)$

(6-g)



overflow occur:

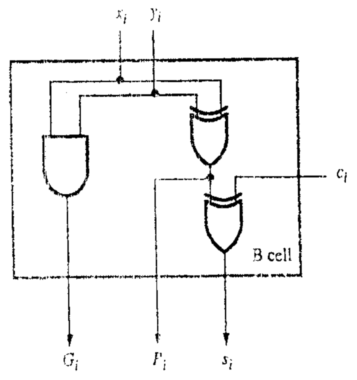
When $\Rightarrow$ x_{n-1} & y_{n-1} are the same

1 0 → carry ← 0 1

-ve $\boxed{1}$	+ve $\boxed{0}$
-ve $\boxed{1}$	+ve. $\boxed{0}$
overflow → $\boxed{0}$	$\boxed{1}$ ← overflow

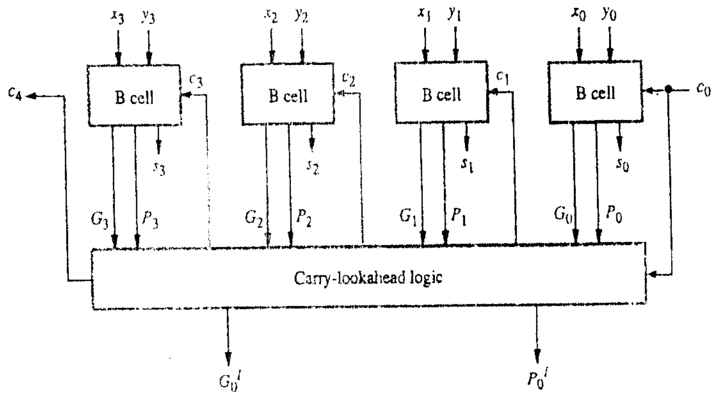
C_{n-1}	C_n	$C_{n-1} \oplus C_n$
0	0	0
0	1	1
1	0	1
1	1	0

} ⇒ overflow



(a) Bit-stage cell

f_i



(b) 4-bit adder

Figure 6.4 4-bit carry-lookahead adder.

Continuing this type of expansion, the final expression for any carry variable is

$$c_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \cdots + P_iP_{i-1}\cdots P_1G_0 + P_iP_{i-1}\cdots P_0c_0 \quad [6.1]$$

Thus, all carries can be obtained three gate delays after the input signals X, Y, and c_0 are applied because only one gate delay is needed to develop all P_i and G_i signals,

— 6 —

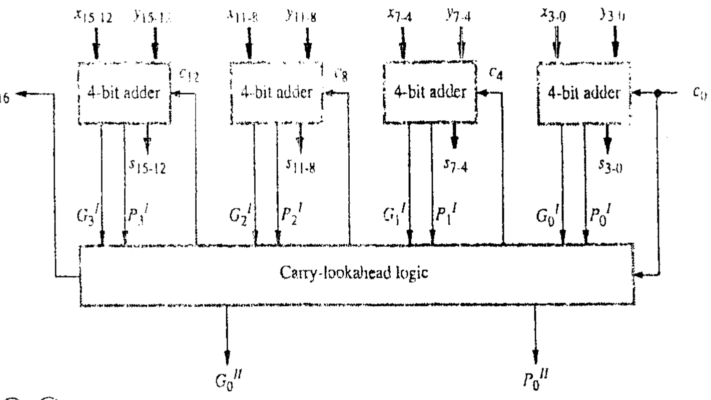


Figure 6.5 16-bit carry-lookahead adder built from 4-bit adders (see Figure 6.4*b*).

Figure 6.5 shows a 16-bit adder built from four 4-bit adder blocks. These blocks provide new output functions defined as G_k^I and P_k^I, where $k = 0$ for the first 4-bit block, as shown in Figure 6.4*b*, $k = 1$ for the second 4-bit block, and so on. In the first block,

$$P_0^I = P_3P_2P_1P_0$$

and

$$G_0^I = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

In words, we say that the first-level G_i and P_i functions determine whether bit stage i generates or propagates a carry, and that the second-level G_k^I and P_k^I functions determine whether block k generates or propagates a carry. With these new functions available, it is not necessary to wait for carries to ripple through the 4-bit blocks. Carry c_{16} is formed by one of the carry-lookahead circuits in Figure 6.5 as

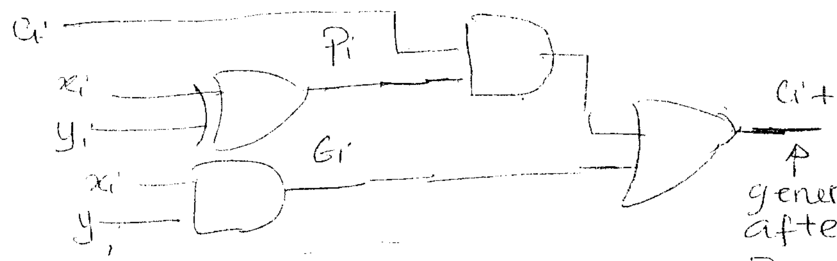
$$c_{16} = G_3^I + P_3^IG_2^I + P_3^IP_2^IG_1^I + P_3^IP_2^IP_1^IG_0^I + P_3^IP_2^IP_1^IP_0^Ic_0$$

The input carries to the 4-bit blocks are formed in parallel by similar shorter expressions. These expressions for c_{16}, c_{12}, c_8, and c_4, are identical in form to the expressions for c_4, c_3, c_2, and c_1, respectively, implemented in the carry-lookahead circuits in Figure 6.4*b*. Only the variable names are different. Therefore, the structure of the carry-lookahead circuits in Figure 6.5 is identical to the carry-lookahead circuits in Figure 6.4*b*. We should note, however, that the carries c_4, c_8, c_{12}, and c_{16}, generated internally by the 4-bit adder blocks, are not needed in Figure 6.5 because they are generated by the higher-level carry-lookahead circuits.

Now, consider the delay in producing outputs from the 16-bit carry-lookahead adder. The delay in developing the carries produced by the carry-lookahead circuits is

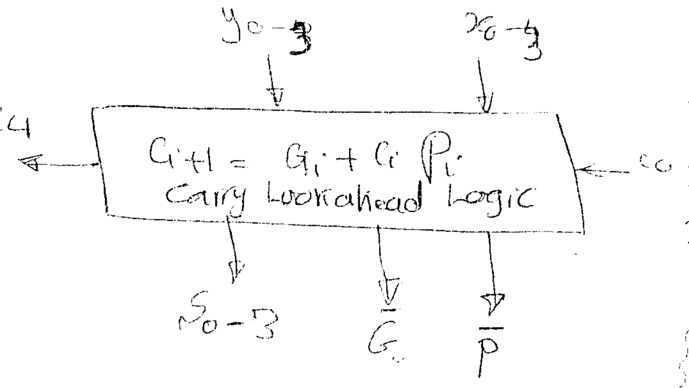
Review

6.10



generated after 3 gate delays

4-bit with carry Lookahead:



(s Gate delay) $\Leftarrow C_4$

fain IN problem

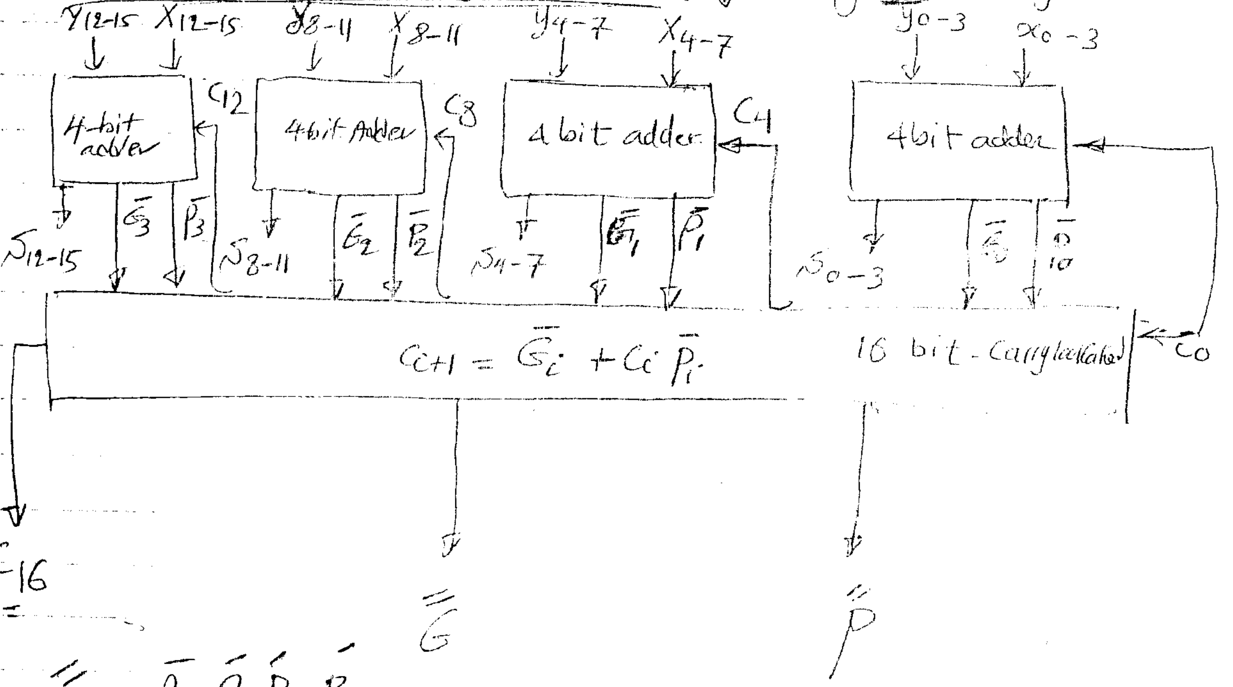
$\bar{P} = P_0 P_1 P_2 P_3$

$\bar{G} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$

$C_1, C_2, C_3, C_4 \Rightarrow$ all are generated after 3 gate delays

16-bit with Carry Lookahead:

using 4-bit carry Lookahead.



$\bar{\bar{P}} = \bar{P}_0 \bar{P}_1 \bar{P}_2 \bar{P}_3$

$\bar{\bar{G}} = \bar{G}_3 + \bar{P}_3 \bar{G}_2 + \bar{P}_3 \bar{P}_2 \bar{G}_1 + \bar{P}_3 \bar{P}_2 \bar{P}_1 \bar{G}_0$

Review

4-bit adder
carry Look ahead

$C_{i+1} = G_i + C_i P_i \rightarrow ①$

$\boxed{\text{Generate:}}$

C_1, C_2, C_3, C_4 at the same time using equation① → 3 gate delays

Generate $\bar{P}_i$, $\bar{G}_i$

$C_1 = G_0 + P_0 C_0$
$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$
$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

$P_i = x_i \oplus y_i$
$G_i = x_i y_i$

$\bar{P}_i = P_0 P_1 P_2 P_3$
$\bar{G}_i = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$

$\bar{P}_i$ → available after 1 gate delay.

$\acute{G}_i$ → available after 2 gate delay.

— 9 —

16-bit - adder
Carry Look ahead
using 4 x 4bit adder.

$C_{i+1} = \underbrace{\underset{2}{\bar{G}_i} + C_i \underset{1}{\bar{P}_i}}_{1} = 5 \rightarrow ②$

$\boxed{\text{Generate:}}$

C_4, C_8, C_{12}, C_{16} all at the same time from equation ② → 5 gate delays

$C_4 = \bar{G}_0 + \bar{P}_0 C_0$
$C_8 = \bar{G}_1 + \bar{P}_1 \bar{G}_0 + \bar{P}_1 \bar{P}_0 C_0$
$C_{12} = \bar{G}_2 + \bar{P}_2 \acute{G}_1 + \acute{P}_2 \acute{P}_1 \acute{G}_0 + \acute{P}_2 \acute{P}_1 \acute{P}_0 C_0$
$C_{16} = \acute{G}_3 + \acute{P}_3 \acute{G}_2 + \acute{P}_3 \acute{P}_2 \acute{G}_1 + \acute{P}_3 \acute{P}_2 \acute{P}_1 \acute{G}_0 + \acute{P}_3 \acute{P}_2 \acute{P}_1 \acute{P}_0 C_0$

$P''_i = \acute{P}_0 \acute{P}_1 \acute{P}_2 \acute{P}_3$
$G''_i = \acute{G}_3 + \acute{P}_3 \acute{G}_2 + \acute{P}_3 \acute{P}_2 \acute{G}_1 + \acute{P}_3 \acute{P}_2 \acute{P}_1 \acute{G}_0$

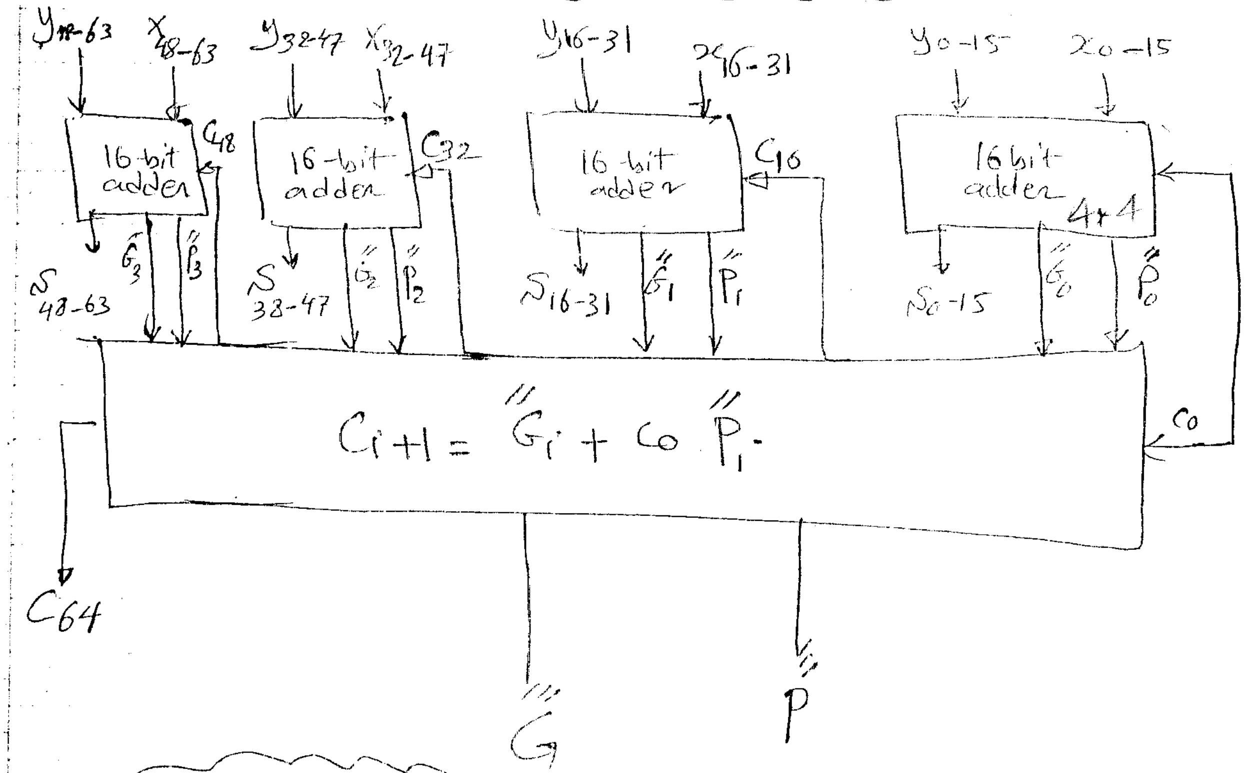
P''_i → need $3 + 1 = 4$ gate delays

G''_i → need $3 + 2 = 5$ gate delays

∴ Both P''_i & G''_i will be available after 5 gate delays.

(a)

64-bit adder using 4*16 bit adder



$C_{i+1} = G''_i + C_0 P''_i$ $\Rightarrow$ will generate $C_{16}, C_{32}, C_{48}, C_{64}$ By the same way as previous.

$C_{16} = G''_0 + P''_0 C_0$

$C_{32} = G''_1 + P''_1 G''_0 + P''_1 P''_0 C_0$

$C_{48} = G''_2 + P''_2 G''_1 + P''_2 P''_1 G''_0 + P''_2 P''_1 P''_0 C_0$

$C_{64} = G''_3 + P''_3 G''_2 + P''_3 P''_2 G''_1 + P''_3 P''_2 P''_1 G''_0 + P''_3 P''_2 P''_1 P''_0 C_0$

all delays need:

G''_i, P''_i $\longrightarrow$ 5 gate delays.

$C_0 P''_i$ $\longrightarrow$ 1 gate delays.

$G''_i + C_0 P''_i$ $\longrightarrow$ 1 gate delays

$C_{16}, C_{32}, C_{48}, C_{64}$ generate after $\longrightarrow$ 7 gate delay.

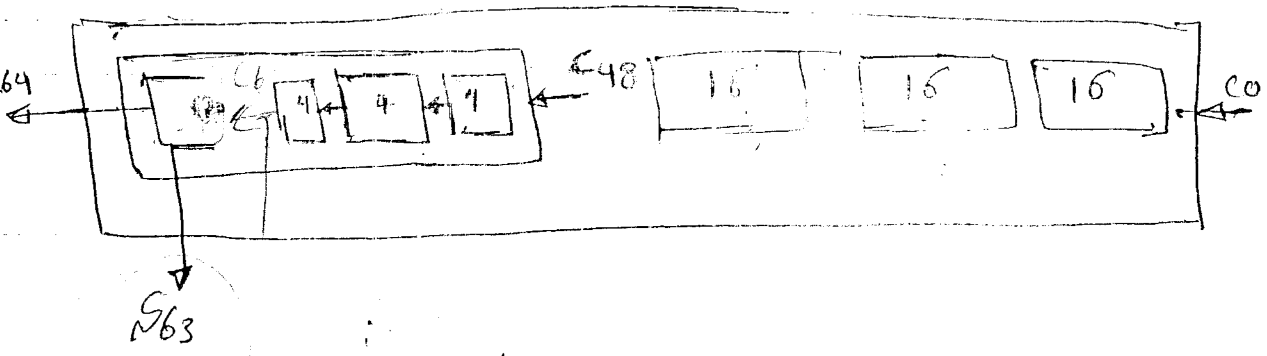
(b) as calculate previous in (a)

C_{64} = 7 gate delays

also C_{16}, C_{32}, C_{48} → generated after 7 gatedelay.

to calculate delay for S_{63}

inside the Last Block



$C_{48} \xrightarrow{\text{produced after}}$ 7 gate delay.

$C_{60} \xrightarrow[\text{into the Last 1 bit adder}]{\text{produced after}}$ 2 more gate delays + 7 = 9 السابقة

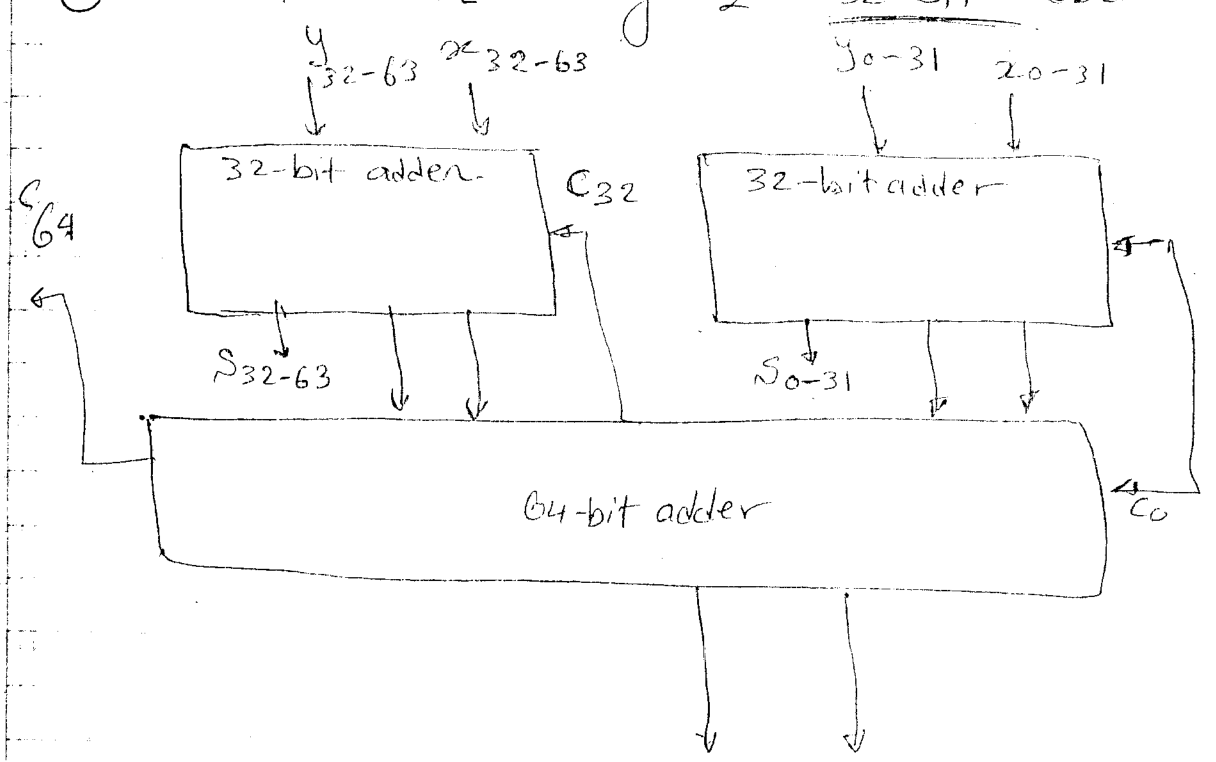
$C_{63} \xrightarrow[\text{inside the 4-bit adder}]{\text{produced after}}$ 2 more gate delays + 9 = 11 السابق

$S_{63} = x_{63} \oplus y_{63} \oplus C_{63}$ → another 1 + gate delay for XOR = $\underline{\underline{12}}$

$= P_{63} \oplus C_{63}$

- 11 -

ⓒ 64-bit adder using 2 * 32-bit adder.



→ Variables S_{31} & C_{32} produced after.

from ⓐ & ⓑ be :-

C_{32} → will x produced after 7 gate delay.

S_{31} → will be produced after 12 gate ~.

as S_{63}

from 64 using 2 * 32-bit adder cascaded 16-bit

C_{32} → produced after 7 gate delays.

S_{31} → after 10 gate delays.

Section 6.2.

Problem $\boxed{6.11}$ Figure 6.4 (a) → 6 [illegible]

(a) # of gates needed to build $\boxed{4\text{-bit}}$ Carry-lookahead:

From Figure 6.4 :-

	# of Unit	# of logic gates:	
B-Cell →	4	3	12 = 4*3
C_1		2	2
C_2		3	3
C_3		4	4
C_4		$\boxed{5}$	5
$\bar{P}_0 = P_0P_1P_2P_3$		1	1
$\bar{G}_0 = \dots$		4	4
Total gate delays			$\boxed{31}$

19 Carry lookahead

[illegible] 6.10 g

(b) # of gates required for 16 bit using 4 * 4-bit

- We need 4 * 4bit-adder (a) = 4 * 31 = 128
- Carry-lookahead Logic need = 19 كما تم بالأعلى

total Logic gates required = 143

- We need to subtract logic gates needed for $\boxed{C_4, C_8, C_{12}, C_{16}}$ ⇒ تم حسابهم مرتين في Carry Lookahead

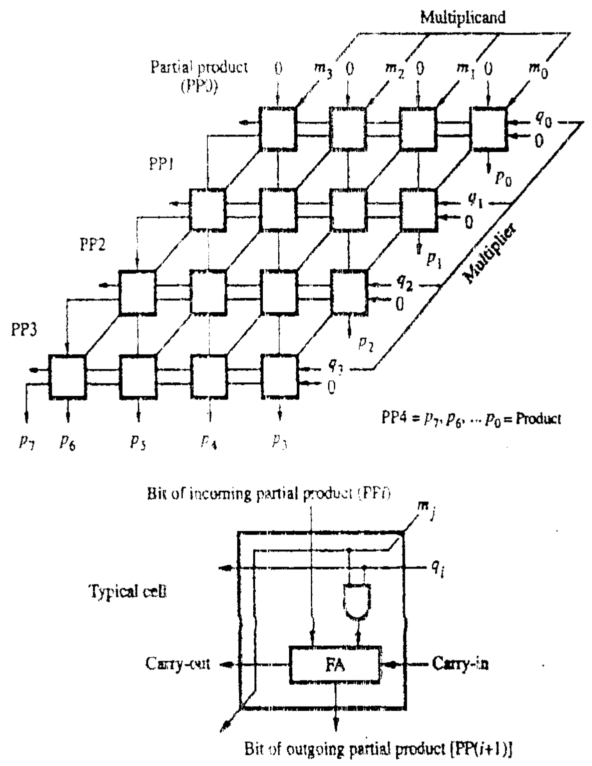
C_8 ، C_4 الذي تم أخذه logic gates $\underbrace{5 \times 4}$ = ٢٠

- Total needed gates = 143 - 20 = 123 gates

-13-   2/9   [illegible]

```
          1 1 0 1     (13) Multiplicand M
        × 1 0 1 1     (11) Multiplier Q
        ---------
          1 1 0 1
        1 1 0 1
      0 0 0 0
    1 1 0 1
  ---------------
  1 0 0 0 1 1 1 1     (143) Product P
```

(a) Manual multiplication algorithm



(b) Array implementation

Figure 6.6 Array multiplication of positive binary operands.

-14-

problem $\boxed{6.12}$ ضرب Sequential Multiplication

$$d(n) = 6(n-1) - 1$$

ex $\Rightarrow$ $d(4) = 6*(3-1) - 1 = 17$ gate delay

for :

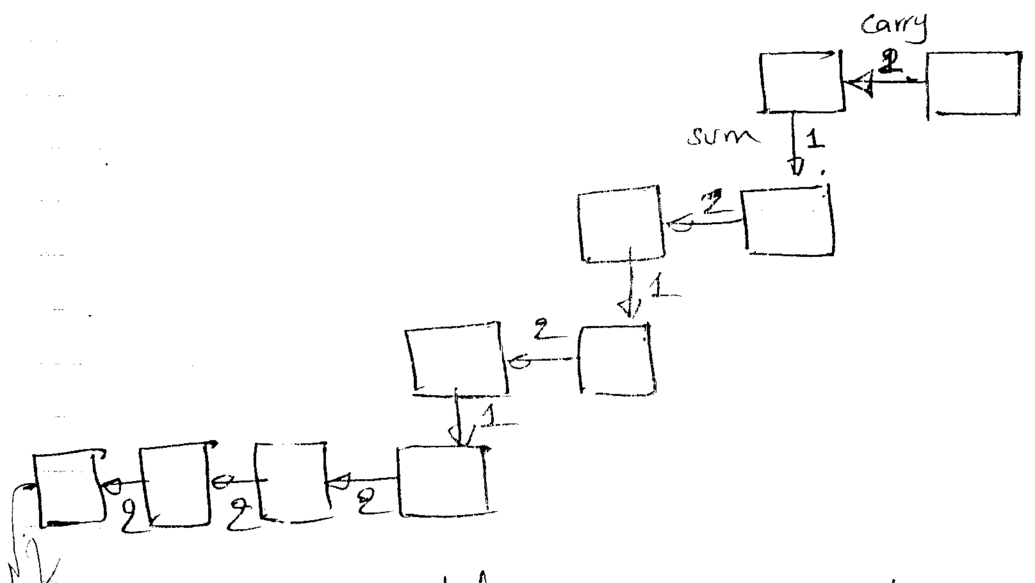
فرض

1 $\rightarrow$ first row $= 1$
(n-2) $\rightarrow$ other rows $= (n-2) * 4$
n $\rightarrow$ Last row $= 2n$

assume
$d(\text{carry}) = 2$
$d(\text{sum}) = 2$
using 2 xor

total delay needed $= 1 + 4(n-2) + 2n$
$= 1 + 4n - 8 + 2n = 6n - 7$

$d(n) = 6n - 6 - 1 = 6(n-1) - 1$



delay

total delays $= 2 + 1 + 2 + 1 + 2 + 1 + 2 + 2 + 2 + 2 = 17$ gate delays

M 00101 ×
Q 10101

A Q

6.14. The multiplication and division charts are:

A × B :

M $\boxed{00101}$

C	A	Q	
$\boxed{0}$	$\boxed{00000}$	$\boxed{10101}$	Initial configuration
0 0	00101 00010	10101 11010	1st cycle
0 0	00010 00001	11010 01101	2nd cycle
0 0	00110 00011	01101 00110	3rd cycle
0 0	00011 00001	00110 10011	4th cycle
0 0	00110 00011	10011 01001	5th cycle

$\underbrace{00011\ 01001}_{\text{product}}$

Restoring division

000101 ⟌ 10101
M Q

00 000 A

A / B :

```
                 [000000]   [10101]
                    A          Q
                 [000101]                      Initial configuration
                    M

shift            000001     0  1  0  1  [ ]
subtract         111011                        1st cycle
                 111100     0  1  0  1  [0]

shift            111000     1  0  1 [0] [ ]
add              000101                        2nd cycle
                 111101     1  0  1 [0] [0]

shift            111011     0  1 [0] [0] [ ]
add              000101                        3rd cycle
                 000000     0  1 [0] [0] [1]

shift            000000     1 [0] [0] [1] [ ]
subtract         111011                        4th cycle
                 111011     1 [0] [0] [1] [0]

shift            110111    [0] [0] [1] [0] [ ]
add              000101                        5th cycle
                 111100    [0] [0] [1] [0] [0]
                           └────────────────┘
add              000101         quotient
                 000001
                 ──────
                remainder
```

note :-
```
 0 = 0 ← 0
-1 = 1 ← 0     (2's M)
+1 = 0 ← 1
 0 = 1 ← 1
```

6.17. The multiplication answers are:

```
(a)    010111       +23                       0 1 0 1 1 1
     × 110110     ×  -10                  ×   0 -1+1 0 -1 0
                  ------                  -------------------
                   -230                                     0
                             sign   [  1 1 1 1 1 1 0 1 0 0 1
                          extension |                     0
                                    |  0 0 0 0 1 0 1 1 1
                                    [  1₁1₁1₁0₂1₁0₁0₁1
                                    ---------------------------
                                       1 1 1 1 0 0 0 1 1 0 1 0

(b)    110011       -13                       1 1 0 0 1 1
     × 101100     ×  -20                  ×  -1+1 0 -1 0 0
                  ------                  -------------------
                    260                                     0
                                                          0
                             sign   [  0 0 0 0 0 0 1 1 0 1
                          extension |                   0
                                    |  1 1 1 1 0 0 1 1
                                    [  0₁0₁0₁1₁1₁0₂1₁
                                    ---------------------------
                                       0 0 0 1 0 0 0 0 0 1 0 0

(c)    110101       -11                       1 1 0 1 0 1
     × 011011     ×   27                  ×  +1 0 -1+1 0 -1
                  ------                  -------------------
                   -297             [  0 0 0 0 0 0 0 0 1 0 1 1
                                    |                       0
                             sign   |  1 1 1 1 1 1 0 1 0 1
                          extension |  0 0 0 0 0 1 0 1 1
                                    |                 0
                                    [  1₁1₁1₁0₁1₁0₁1₁ ₁
                                    ---------------------------
                                       1 1 1 0 1 1 0 1 0 1 1 1

(d)    001111        15                       0 0 1 1 1 1
     × 001111     ×  15                   ×   0+1 0 0 0 -1
                  ------                  -------------------
                    225                1 1 1 1 1 1 1 1 0 0 0 1
                                       0 0 0 0 1 1 1 1
                                    ---------------------------
                                       0 0 0 0 1 1 1 0 0 0 0 1
```

Both alg

```
0 1 1 0 1 1 0
+1 0 -1 +1 0 -1
 ↓     ↓     ↓
+2    -1    -1
```

+1 → M

-1 → 2's M

+2 → [illegible] M

-2 → [illegible] M

and [illegible] M

0 → 0[illegible]

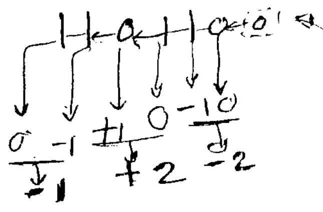
to shift left

-17-

$\xleftarrow{}$ كاسر $\boxed{0}$

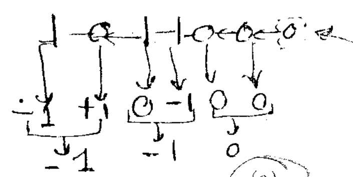
6.18. The multiplication answers are:

(*a*) 010111 × 110110



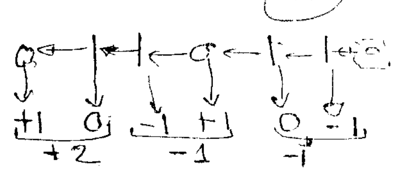
```
                 0 1 0 1 1 1
                  -1  +2  -2
 1 1 1 1 1 1 0 1 0 0 1 0
 0 0 0 0 1 0 1 1 1 0
 1 1 1 1 0 2 1 1 0 1 0 1 1
 ------------------------
 1 1 1 1 0 0 0 1 1 0 1 0
```

(*b*) 110011 × 101100



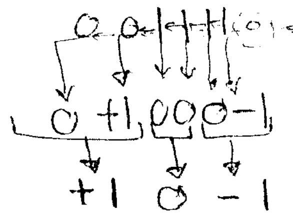
```
                 1 1 0 0 1 1
                  -1  -1   0
                           0
 0 0 0 0 0 0 1 1 0 1
 0 0 0 0 1 1 1 1 1 0 1 1
 ------------------------
 0 0 0 1 0 0 0 0 0 1 0 0
```

(*c*) 110101 × 011011



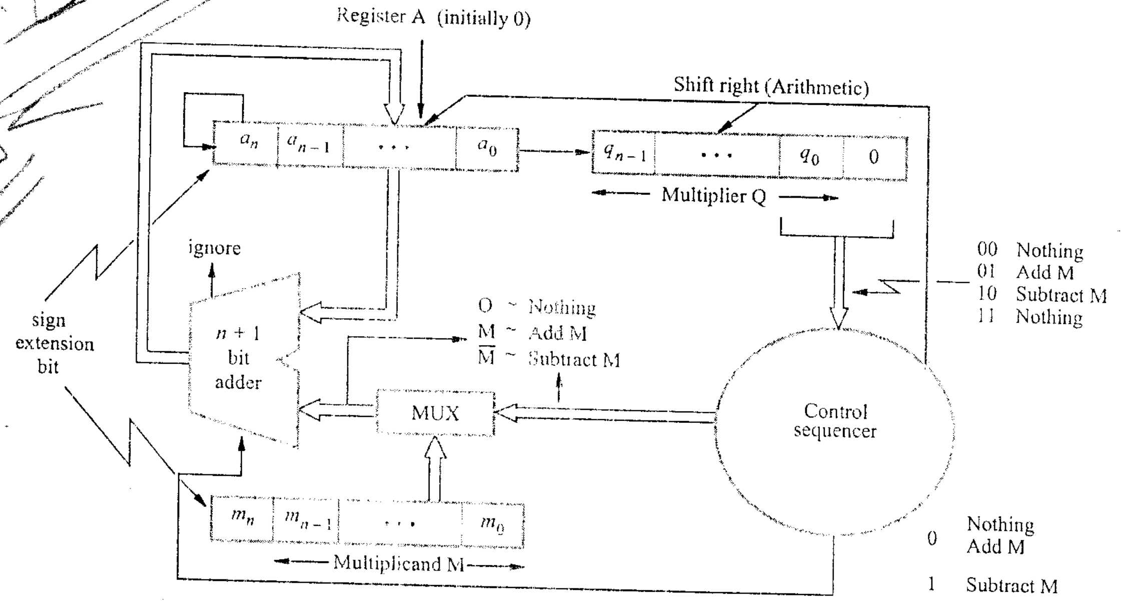
```
                 1 1 0 1 0 1
                  +2  -1  -1
 0 0 0 0 0 0 0 0 1 0 1 1
 0 0 0 0 0 0 1 0 1 1
 1 1 1 0 1 0 1 1
 ------------------------
 1 1 1 0 1 1 0 1 0 1 1 1
```

(*d*) 001111 × 001111



```
                 0 0 1 1 1 1
                  +1      -1
 1 1 1 1 1 1 1 1 0 0 0 1
 0 0 0 0 1 1 1 1
 ------------------------
 0 0 0 0 1 1 1 0 0 0 0 1
```

6.19 Both the A and M registers are augmented by one bit to the left to hold a sign extension bit. The adder is changed to an $n + 1$-bit adder. A bit is added to the right end of the Q register to implement the Booth multiplier recoding operation. It is initially set to zero. The control logic decodes the two bits at the right end of the Q register according to the Booth algorithm, as shown in the following logic circuit. The right shift is an arithmetic right shift as indicated by the repetition of the extended sign bit at the left end of the A register. (The only case that actually requires the sign extension bit is when the n-bit multiplicand is the value $-2^{(n-1)}$; for all other operands, the A and M registers could have been n-bit registers and the adder could have been an n-bit adder.)



-19-

www.elsolucionario.net